Использование proof assistants для описания операционных семантик

Кирилл Таран

445 группа

15 апреля 2013 г.

науч. рук.: Булычев Д.Ю.

О проекте

- Исследование в рамках лаборатории JetBrains
- Области исследования:
  - разработка языков программирования
  - сертификационное программирование
    - К программе прилагаются “сертификаты” – формальные описания и доказательства её полезных свойств.

Введение

Proof assistants

- Сертификационное программирование появилось в связи с развитием т.н. proof assistants – систем для интерактивного построения доказательств теорем, автоматической проверки доказательств и т.д.
- В данной работе рассмотрены системы, основанные на зависимых типах

Введение

Зависимые типы

- “Зависимые типы” – могут быть параметризованы термами (значениями), в отличие от простых полиморфных типов, которые могут быть параметризованы только другими типами
- Язык программирования с зависимыми типами позволяет работать с теоремами (Curry–Howard correspondence):
  - формулировка теоремы – тип функции
  - терм, определяющий функцию – доказательство

Цели работы

- Операционная семантика – дедуктивная система, поэтому инструменты для доказательства теорем могут оказаться полезными при работе с ней
- Цель курсовой – оценить применимость р.а. к задачам разработки языков программирования
- Предполагаемые результаты:
  - автоматическая генерация интерпретатора по операционной семантике
  - доказательство каких-либо свойств об интерпретации (по возможности автоматическое)
  - рассмотреть различные системы (по возможности)

Операционные семантики

Семантическое дерево

- Семантика задаёт переходы между состояниями
- Программе соответствует семантическое дерево – дерево применений правил семантики к синтаксическим элементам программы
- Пример дерева:

  программа ‘(z:=x ; x:=y) ; y:=z’

$$\dfrac{\dfrac{\langle z := x, s_0\rangle \to s_1 \qquad \langle x := y, s_1\rangle \to s_2}{\langle z := x; x := y, s_0\rangle \to s_2} \qquad \langle y := z, s_2\rangle \to s_3}{\langle (z := x; x := y); y := z, s_0\rangle \to s_3}$$

  s_0, s_1, s_2 – состояния программы (списки значений)

Операционные семантики

Правила вывода (big-step/natural semantics)

";"	$\dfrac{\langle p_1, s_0\rangle \to s_1 \qquad \langle p_2, s_1\rangle \to s_2}{\langle p_1; p_2, s_0\rangle \to s_2}$
"if-true"	$s_0[b] = true \Rightarrow \dfrac{\langle p_1, s_0\rangle \to s_1}{\langle \text{if b then } p_1 \text{ else } p_2, s_0\rangle \to s_1}$
":="	$\langle x := a, s\rangle \to s[x \mapsto a]$

Операционная семантика на Agda

Тип данных

- Семантика представляется на языке Agda в виде типа данных:
  - Термы типа – семантические деревья
  - Параметры типа – входные/выходные состояния перехода и программа
  - Конструкторы – правила перехода

```
data Transition (s₁ : State) : S → State → Set where
  [skip]   :   Transition s₁   skip   s₁
  [assign] : ∀ {k v} → Transition s₁ (assign k v) ((k , s₁ [[ v ]]) |> s₁)
  [comp]   : ∀ {s₂ s₃ p₁ p₂} → Transition s₁   p₁   s₂
                              → Transition s₂   p₂   s₃
                              → Transition s₁ (comp p₁ p₂) s₃

-- et cetera ...
```

Операционная семантика на Agda

Тип данных

- Преимущества подобного представления:
  - Задаёт отношение “вычислимости”, которое можно использовать в типах функций
  - Кроме результата вычисления интерпретатор будет строить дерево вычисления, которое можно использовать для анализа интерпретатора
  - Интерпретатор можно попытаться вывести автоматически

```
data Transition (s₁ : State) : S → State → Set where
  [skip]   :   Transition s₁   skip   s₁
  [assign] : ∀ {k v} → Transition s₁ (assign k v) ((k , s₁ [[ v ]]) |> s₁)
  [comp]   : ∀ {s₂ s₃ p₁ p₂} → Transition s₁   p₁   s₂
                              → Transition s₂   p₂   s₃
                              → Transition s₁ (comp p₁ p₂) s₃

-- et cetera ...
```

Операционная семантика на Agda

Интерактивное построение интерпретатора

- Код, введённый вручную:

```
-- Interpretation s₁ p = (s₂ : State, Transition s₁ p s₂)
interpret : (s : State) → (p : S) → Interpretation s p

-- {!!} - “hole”, выводится автоматически
interpret s₁ (skip    ) = {!!}
interpret s₁ (assign   k   v ) = {!!}
interpret s₁ (comp     p₁ p₂) with interpret s₁ p₁
... | ⟨ s₂ , tr₁ ⟩ with interpret s₂ p₂
... | ⟨ s₃ , tr₂ ⟩ = {!!}
-- et cetera ...
```

- Сгенерированный код:

```
-- Interpretation s₁ p = (s₂ : State, Transition s₁ p s₂)
interpret : (s : State) → (p : S) → Interpretation s p
interpret s₁ (skip    ) = ⟨ s₁ , [skip] ⟩
interpret s₁ (assign   k   v ) = ⟨ (k , (s₁ [[ v ]])) |> s₁ , [assign] ⟩
interpret s₁ (comp     p₁ p₂) with interpret s₁ p₁
... | ⟨ s₂ , tr₁ ⟩ with interpret s₂ p₂
... | ⟨ s₃ , tr₂ ⟩ = ⟨ s₃ , [comp] tr₁ tr₂ ⟩
-- et cetera ...
```

Операционная семантика на Agda

Интерактивное построение интерпретатора

- Автоматический вывод термов решается с помощью Agsy – компонента Agda для поиска type inhabitant (auto proof search)
- Agsy пытается разобрать по случаям аргументы функции и применить данные ему подсказки (леммы), при этом проверяя типизируемость генерируемого выражения
- К сожалению, есть ограничения реализации:
  - сопоставление с образцом произвольных подвыражений Agsy выводить не умеет – пока что приходится писать вручную
  - но возможно получится обойтись функциями специального вида для разбора случаев

Операционные семантики

Правила вывода (small-step/structural semantics)

";"	$\dfrac{\langle p_1, s_0\rangle \to s_1}{\langle p_1; p_2, s_0\rangle \to s_1\ \underline{p_2}}$ $\dfrac{\langle p_1, s_0\rangle \to s_1\ \underline{p_1'}}{\langle p_1; p_2, s_0\rangle \to s_1\ \underline{p_1'; p_2}}$
"if-true"	$s[b] = \mathit{true} \Rightarrow \langle \text{if b then } p_1 \text{ else } p_2, s\rangle \to s\ \underline{p_1}$
":="	$\langle x := a, s\rangle \to s[x \mapsto a]$

Операционная семантика на Agda

Интерактивное доказательство свойств

- Теорема об эквивалентности big-step и small-step семантик:

$$
\begin{aligned}
&\text{sos}{\equiv}\text{ns} : (s : \text{State}) \to (p : \text{S}) \to \forall\ \{s\text{-}sos\ s\text{-}ns\} \\
&\quad \to (tr\text{-}sos : \text{Transition-SS*}\ s\ p\ s\text{-}sos) \\
&\quad \to (tr\text{-}ns : \text{Transition-NS}\ s\ p\ s\text{-}ns) \\
&\quad \to s\text{-}sos \equiv s\text{-}ns \\
&\text{sos}{\equiv}\text{ns}\ s\ p\ \{s\text{-}sos\}\ \{s\text{-}ns\}\ tr\text{-}sos\ tr\text{-}ns = \{!!\}
\end{aligned}
$$

- Кода в доказательстве очень много, без автоматического вывода тяжело

Результаты

- Найден способ полуавтоматического построения интерпретатора и доказательств свойств о нём
- Есть возможность дополнить существующие инструменты до полной автоматизации
- Были также рассмотрены Coq и Idris, использующие тактики вместо auto proof search
  - $\oplus$ тактики дают больше возможностей по управлению выводом
  - $\ominus$ но требуют работы по написанию proof script
- Репозиторий с кодом:
  - https://github.com/kirillt/exercises/tree/master/semantics (наброски семантик на Coq, Agda, Idris + интерпретатор на Agda)